Developing Ambient Intelligence Systems: A Solution based on Web Services

Valérie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Françoise Sailhan, Rafik Chibout, Nicole Levy, Angel Talamona

Developing Ambient Intelligence Systems: A Solution based on Web Services*

Valérie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Françoise Sailhan, Rafik Chibout, Nicole Levy, Angel Talamona

ARLES Research Team,
INRIA-Rocquencourt, France
http://www-rocq.inria.fr/arles

Abstract. Enabling the ambient intelligence vision means that consumers will be provided with universal and immediate access to available content and services, together with ways of effectively exploiting them. Concentrating on the software system development aspect, this means that the actual implementation of any ambient intelligence application requested by a user can only be resolved at runtime according to the user's specific situation. This paper introduces a base declarative language and associated core middleware, which supports the abstract specification of Ambient Intelligence applications together with their dynamic composition according to the environment. The proposed solution builds on the Web services architecture, whose pervasiveness enables both services availability in most environments, and specification of applications supporting automated retrieval and composition. In addition, dynamic composition of applications is dealt in a way that enforces the quality of service of deployed applications in terms of security and performance.

Keywords. Ambient intelligence, declarative language, middleware, mobile computing, quality of service, service location, Web services.

1 Introduction

The vision of *ubiquitous computing* that was introduced in the early 90s [Weiser, 1993] is now giving rise to extensive research [Dertouzos, 1999, Esler et al., 1999, IST Advisory Group, 2001, Garlan et al., 2002, Milojicic et al., 2001]. While ubiquitous computing could be considered as quite futuristic in the early 90s, the combined effect of advances in the areas of hardware and network technologies, and of the universal usage of Internet-based technologies and wireless phones makes ubiquitous computing almost a reality. The vision is now termed *ambient intelligence* or *pervasive computing* to emphasize that it does not solely rely on ubiquitous computing (i.e., useful, pleasant and unobtrusive presence of computing devices everywhere) but also on *ubiquitous networking* (i.e., access to network and computing facilities everywhere) and

*Revised version to appear in "Journal of Automated Software Engineering"

on *intelligent aware interfaces* (i.e., perception of the system as intelligent by people who naturally interact with the system that automatically adapts to their preference). The distinction between ambient intelligence and pervasive computing systems relates to the end-users who are more specifically targeted: ambient intelligence systems are aimed at consumers and are hence oriented towards infotainment applications, while pervasive computing systems target professional users and thus desktop applications [Aarts et al., 2001]. This paper concentrates more specifically on ambient intelligence enabling, although the proposed approach applies as well to pervasive computing.

While available technologies provide us with base enablers of the ambient intelligence vision, there is still a long way to go before offering robust ambient intelligence systems to consumers, requiring advances in most areas relating to the computer science field (e.g., hardware supporting low-power high-performance wireless devices, network ensuring connectivity everywhere, human-computer interaction enabling intelligent multi-modal interfaces, development support for deploying ubiquitous applications). This paper addresses support for the development of ambient intelligence software systems, concentrating more specifically on enabling seamless access to content and services any-time, any-where. Our solution lies in a base declarative language for specifying ambient intelligence software systems and associated core middleware infrastructure for any-time, any-where access (i.e., ubiquitous computing), which further builds upon available network technologies for enhanced connectivity (i.e., ubiquitous networking). Offering ambient intelligence software systems then relies on the development of base services implementing functionalities related to applications, interfaces and middleware. The key feature of our solution relates to enabling the dynamic composition, possibly distributed, of requested services (i.e., functions provided by the computing system) according to the mobile users' situation, while guaranteeing quality of service to users in terms of at least performance and security properties. Our solution further builds upon the Web services architecture [W3C, 2002] that is pervasive enough for guaranteeing services availability in most environments and consistent specification of composable services.

We more specifically complement the Web services architecture with support dedicated to the *situation-sensitive composition* of Web services, i.e., the dynamic selection and composition of (application- and middleware-related) Web services for serving the users' requests while offering quality of service. Currently, the main constituents of the Web services architecture are [W3C, 2002]: (i) WSDL (Web Services Description Language) that is a declarative language for describing the interfaces of Web services; and (ii) SOAP (Simple Object Access Protocol) that defines a lightweight protocol for information exchange. The Web services architecture is further conveniently complemented by UDDI (Universal Description, Discovery and Integration) that is a specification of a registry for dynamically locating and advertising Web services (see http://www.uddi.org). Given the Web services architecture, our solution lies in the WSAMI (Web Services for AMbient Intelligence) environment that decomposes into:

- A declarative language, based on existing Web services languages for the specification of both application- and middleware-related services. Given the WSAMI specification of a service, an instance is automatically selected and composed upon a user request, according to the services that may be retrieved in the environment. WSAMI specification in particular allows for dynamic, distributed service composition over the nodes hosting

the integrated services, including customization of the network links for enforcing quality of service.

- A core SOAP-based middleware realizing situation-sensitive composition of services, which offers a *naming&discovery service* that supports naming, discovery and lookup of services, both in the local- and wide-area over WLAN (Wireless Local Area Network). The core middleware handles requests for any Web service characterized by its WSAMI abstract interface, seeking matching instances of related services in the environment. In general, ambient intelligence systems will rely on a large number of middleware-related services dealing with, e.g., intelligent aware user interfaces, content filtering, caching, security, etc. The set of middleware-related services is left open and available ones will be exploited via situation-sensitive composition. The only requirement is for the services to be specified in the WSAMI language and made available through the core *naming&discovery service*.

The next section surveys existing work in the area of mobile distributed systems development, addressing in particular the benefit of introducing an approach based on Web services for the development of ambient intelligence software systems. Section 3 then provides an overview of WSAMI, further illustrating the exploitation of situation-sensitive composition of Web services with respect to typical ambient intelligence scenarios. Section 4 details the WSAMI language for the specification of Web services, which specifically enables their situation-sensitive composition in the mobile context. As presented in Section 5, actual situation-sensitive composition at runtime relies on the WSAMI minimal core middleware infrastructure, i.e, SOAP-based core broker enriched with a naming&discovery service, which must be run on any terminal that is willing to take part in ambient intelligence applications. Section 6 then presents a prototype implementation and evaluation of the WSAMI environment. Finally, Section 7 assesses our contribution with respect to related work and discusses our current and future work.

2 Background

Existing work on the dynamic composition of distributed services relates to the design of mobile systems whose main objective is to assist the development of services in the presence of mobility.

2.1 Mobile Computing Systems

The term mobility is quite broad and may be subdivided into three categories [Roman et al., 2000]: (i) personal mobility that deals with the mobility of people who may not necessarily carry a device, (ii) computer mobility that deals with the mobility of devices, and (iii) computational mobility that deals with the migration of code over physical nodes. The two first are inherent features of ambient intelligence systems while the third is a software technology mean that has been proven useful for dealing with distributed computing in low-bandwidth environments (e.g., Internet computing but applicable as well to wireless computing) and with on-line upgrading of the local computing environment. Although computational mobility may be useful for the implementation of ambient intelligence systems, it does not need to be supported in its most general form as enabled by mobile agent systems.

In addition, ambient intelligence systems that would solely be based on the mobile agent technology would require availability of the supporting platform on most of the nodes, which is hardly achievable in an open environment. This further raises the issue of security regarding both protecting host against malicious agents but also agents against malicious hosts. While the former security aspect is now well addressed, the latter is still an open issue.

Personal mobility and computer mobility are often addressed in conjunction, since they are both concerned with enabling access to the computing space (i.e., content and computations, either public or private) from various locations. The only difference lies in the availability of the user's profile for setting the user's environment, which requires special care in the case of personal mobility. However, this is no longer a crucial issue since it may be considered that every user will always carry a personal device enabling its identification (e.g., mobile phone or smartcard). Work on supporting computer mobility is centered around the design of dedicated middleware systems. In particular, there has been extensive study on middleware systems managing the distribution of content over mobile nodes. Results in this area include solutions to the handling of disconnected operations [Ekenstam et al., 2001, Joseph et al., 1995, Kuenning and Popek, 1997, Noble et al., 1997, Petersen et al., 1997], the caching of nomadic data on untrusted servers [Kubiatowicz et al., 2000], the sharing of data in ad hoc networks [Mascolo et al., 2001, Boulkenafed and Issarny, 2003, Sailhan and Issarny, 2003], and the location of mobile content [Castro et al., 2001, Baggio et al., 2001]. Middleware architectures have further been proposed to cope with the realization of distributed services in the mobile environment, dealing in particular with environmental changes in terms of resource availability [Kon et al., 2002], discovery and lookup of services in the local area [Bettstetter and Renner, 2000], and middleware heterogeneity [Grace et al., 2003].

Most of the aforementioned references concentrate on offering a middleware infrastructure with associated APIs, leaving under the responsibility of the developers to exploit provided middleware-related services for their applications. In addition, most of the proposed solutions rely on some specific middleware technology, which limits the computing nodes that can actually participate in the realization of distributed mobile systems. Hence, while existing middleware systems for mobility provide a number of useful functionalities enabling ambient intelligence, they do not support the actual deployment of ambient intelligence systems.

2.2 Dynamic System Composition

From our perspective, a base solution to the development of ambient intelligence systems lies in distinguishing the specification of services from their implementation. Such a solution has already been proven successful by component-based middleware and even more so by architecture-based approaches, for easing the development of distributed information systems. In the ambient intelligence context, the implementation of a given service must further be the *situation-sensitive composition* of the service's specification so that the resulting composition does not solely realize the target functional behavior but also enforces quality of service. Such a facility is enabled by component-based middleware, which allows enforcing required quality of service through the integration of adequate middleware-related services [Issarny et al., 2002]. However, customization of the middleware is addressed at design time, including possible middleware adaptation to deal with environmental changes [Blair et al., 2000]. To realize the am-

bient intelligence vision, situation-sensitive composition must be enabled any-time, any-where and hence must not rely on a priori knowledge of the computing environment, and in particular available service instances. In other words, while existing middleware systems require application developers to specify the instances of (middleware- and application-related) services to be used in the composition of applications, this composition shall be automated with respect to the environment in the ambient intelligence context.

Automated retrieval of component instances making up an application is addressed by the Aura project through task-driven computing [Sousa and Garlan, 2002, Cheng et al., 2002]: a user task is modeled as a software architecture description that specifies the abstract application-related services and connectors to be composed, and the environment takes care of instantiating and adapting the architecture according to available component instances and resources. The abstract modeling of user tasks allows for platform-independence regarding the application-related services that are actually invoked. However, the instantiation and adaptation of architectures still place high demand on the underlying platform; it requires availability of components implementing task manager, service suppliers, context observer and environment manager in every environment. Thus, in a way similar to the Aura approach, we propose a solution that is based on the abstract specification of services for their automated composition. However, unlike Aura, our solution places limited requirements on the environment, i.e., any resource (from tini-scale devices to powerful workstations) that is willing to collaborate in the realization of the ambient intelligence system is only required to provide support for service discovery, as opposed to a rich-full distributed runtime environment.

Dynamic and automated composition of middleware-related services for enforcing quality of service according to the environment is addressed by the CANS infrastructure [Fu et al., 2001], which allows injecting components into the network for dynamically adapting the system to resource characteristics of end devices and network links. However, this solution puts high demand on the underlying platform by, e.g., relying on support for mobile code. In addition, system adaptation is addressed at the level of the overall system configuration by requiring a priori knowledge of the whole network path, and further induces significant resource consumption due to associated computation and communication costs. Hence, this solution is more specifically targeted at customizing access to a given proprietary service from a wireless device that is quite static during service access. We target a more general solution, allowing mobile users to access both private and public, application-related services, possibly composed, while offering quality of service by integrating needed middleware-related services on the communication path in a decentralized way.

2.3 Enabling Ambient Intelligence via the Web

Dynamic integration of middleware-related with application-related services, enabling tuning of service implementation with respect to available resources, depends on the reachability of services within the network. Hence, the achieved quality of service may in the worst case comply with the best effort semantics, but the implementation of services gets achievable in most environments. However, such a feature can only be realized through a technology that is pervasive enough for guaranteeing both (application-related and middleware-related) services availability in most environments, and consistent specification of services for automated

retrieval and composition. Services should further be composable in nature, i.e., built for being integrated with other services *via* adequate interaction in terms of message exchanges. The Web provides us with such a technology.

The universality of XML (Extensible Markup Language) has led to the definition of the Web services architecture [W3C, 2002], which allows interoperation among heterogeneous software systems. The main features of Web services are the following:

- They are accessible on the Web at a given Uniform Resource Identifier (URI).
- They are composable in nature, i.e., a Web service may depend on other Web services, and their composition may be static (at creation time) or dynamic (at runtime).
- They are federated in nature, i.e., interaction between Web services may cross organization boundaries.
- They are heterogeneous in nature, i.e., they may be implemented on a variety of platforms such as J2EE (see http://java.sun.com/j2ee) or .NET (see http://www.microsoft.com/net).
- They agree upon standard exchange protocols and languages to interact with each other, e.g., XML, WSDL, SOAP, UDDI.

Web services have initially been introduced for supporting business processes built out of the composition of services implementing complex business-related applications. However, features of Web services make them also particularly suited for the dynamic composition of systems, as required by ambient intelligence. A Web service is then viewed as a networked service, which may implement either application- or middleware-related functionalities, and be available on any kind of node. In particular, Web services can be made available on most platforms, including tiny-scale devices considering ongoing work in the area of embedded systems [Borriello and Want, 2000]. Latest research in the area further shows increasing interest for making complex Web services available on/to wireless nodes through caching [Terry and Ramasubramanian, 2003, Sairamesh et al., 2002, Friedman, 2002]. In addition, the Semantic Web effort targets the definition of ontologies enabling machine reasoning about the Web content [W3C, 2001], which may be exploited for enforcing consistent specification of services.

Specific requirements imposed by ambient intelligence systems on the Web services architecture relate to supporting the dynamic selection and composition of Web services. This must further be realized in a way that enforces quality of service, while accounting for mobility and resource constraints of wireless devices. The mobility aspect leads us to: (i) address the selection of services instances with respect to the node accessing the services, as opposed to the dynamic computation of the overall service configuration in a central location that is undertaken, e.g., in the CANS infrastructure, and (ii) account for latest advances in wireless technologies and in particular favor the exploitation of WLAN that uses the unlicensed wireless spectrum and appears to be a major enabler of ubiquitous networking, considering in particular the respective advantages of supported ad hoc and infrastructure-based networking modes [Ritter, 2003, McFarland and Wong, 2003, Poor et al., 2003]. The quality of service aspect leads us to deal with the dynamic selection and composition of middleware-related services

for customizing the network path between any two communicating services. The next section introduces our solution to adapting the Web services architecture to meet the requirements of ambient intelligence applications.

3 WSAMI: Mobile Web Services for Ambient Intelligence

The following illustrates the realization of ambient intelligence scenarios using dynamic composition of Web services.

3.1 Enabling Ambient Intelligence Scenarios via Web Services Composition

Our work is part of the effort of the IST Ozone project [OZONE, 2002], which investigates the design and prototype implementation of a first instance of a generic framework enabling consumer-oriented ambient intelligence applications, i.e., allowing consumers to access naturally, freely, any-time, any-where, the multimedia content that is available at home and on the Internet. A number of demonstrator scenarios have been devised to assess the target functionalities with respect to the expectation of consumers, and to further experiment with the project results [OZONE, 2003]. We present here the scenarios that will be enabled by one of the Ozone demonstrators, which has been developed at INRIA on top of the WSAMI middleware.

> The Rocquencourt city, near Versailles, offers CyberCars[1], which are fully automated cars that do not require any driver assistance. CyberCars may be booked via the Internet so that a person visiting Rocquencourt has a CyberCar waiting for him/her when he/she arrives at the specified location in Rocquencourt. All the wireless devices (i.e., the CyberCar computer and the users' PDAs–Personal Digital Assistants) are equiped with a WLAN card (i.e., IEEE 802.11b in the experiment).
>
> Michel waits for his friends (Sylvie, Paul and John) at the reception area of the Rocquencourt tennis club. Michel lives in Rocquencourt, the others in Paris. They have already arranged their game 3 days ago in their common Ozone Sports Schedule.
>
> John and Paul travel to Rocquencourt by public transportation. When they arrive in Rocquencourt, Paul reserves two seats in a Cybercar to the tennis club through speech commands using the kiosk station in Rocquencourt.
>
> Sylvie arrives by car a little bit late. As Rocquencourt downtown is not accessible by car, she calls a Cybercar to take her from the parking lot to the tennis court. When entering the Cybercar, Sylvie's PDA automatically provides the destination to the Cybercar that takes her to the tennis club.
>
> While on their way to the tennis court, John and Paul call Michel on John's PDA and ask him if he can recommend a movie for them to see after the game. Actually, he does. Using his PDA, he sends them the trailer and the cinema's location and schedule, which he already had stored on his personal user space. John and Paul watch the trailer on the Cybercar's screen. Now, they want to listen audio critics

[1] see http://www.cybercars.org

about the movie. But, there is a lot of noise in the street due to major work on the road and the stereo speakers of the Cybercar are not powerful enough, to clearly hear the audio critics. The interaction modality is thus turned into textual interaction. John and Paul finally agree on the movie and after asking Sylvie, they purchase 4 tickets.

There's one thing lacking: where to go for dinner ? John and Paul take a quick review on the Rocquencourt City Ozone Information System and choose a restaurant after checking its menu and doing a 3D virtual tour of its interiors and seeing a video of its chef showing their main specialties. They make reservations.

John, Paul and Michel have just met at the reception area of the tennis club when Sylvie's vehicle arrives. Before starting the match, Michel makes a Cybercar reservation for the trip to the cinema and another for the trip to the restaurant. They play their game, and after the match go to the cinema.

Before the cinema, the four friends make their arrangements for their return. Paul decides to go back to Paris; he accesses the Ozone Transport Information Service in the Cybercar terminal, for the multi-modal transport option (taxi, bus, railway). John is staying in Rocquencourt and thus makes a hotel reservation, as he plans to visit the Versailles castle the next day. After the cinema, they have dinner together. When finishing, John picks up a Cybercar to bring him to his hotel, Sylvie takes another Cybercar to where her car is parked and Michel walks home.

The handling of the mobile users' requests of the above scenarios can all be restated in terms of dynamic composition of Web services according to the users' situation. For instance, a significant part of the scenarios relates to accessing public services from a wireless device (either power-plugged like the CyberCar computer or energy-constrained like the PDA): the CyberCar service for booking and scheduling trips, and the restaurant, hotel, theater, city guide, and multi-modal transport services. These services may be implemented via composition of more primitive Web services, as may be, e.g., the case of the multi-modal transport service. These are traditional Web services, which may be accessed via the Internet. However, they may also be possibly directly accessed via the WLAN (e.g., Cybercar booking service), using adequate service discovery in the local area. Another service composition example is the setting of the tennis game through the common sport schedule that is a collaborative service among a set of Web services, i.e., the (public) Rocquencourt Tennis Club service for booking a court and the (private) individual agenda services of the players. The service is then realized through the composition of the individual services, so as to reach common agreement for the game's time, according to court availability and users preferences The service instances that are actually composed at runtime do not depend on the environment in this specific case; the instances are fixed by the request to the collaborative scheduling service, which identifies the tennis club and the game participants. The service instances are thus identified through their URI, which may be acquired in a number of steps by the user, using in particular some (private and/or public) directory service.

The aforementioned composition of services shall further account for related quality of service. For instance, booking requests to services (e.g., Cybercar reservation) may be required

to be transactional by the user if he/she wants to have guaranteed reservation. Security requirements may also be imposed for the sake of privacy and are mandatory if critical content is exchanged (e.g., credit card information for securing the room booking). The content that is delivered back to the user as a result of a service request may further impose temporal constraints (e.g., video streaming) and/or need to be adapted according to available resources (e.g., display size, bandwidth), user preferences, and surrounding environment (e.g., noise).

The above examples of composition focus on realizing application-related services out of existing Web services. This is straightforward to achieve in a stationary environment with Internet access, using available Web services technology. However, we have here to account for the fact that Web services may be hosted by both stationary and mobile nodes, and that the composition shall be realized so as to offer quality of service to the mobile user according to his/her situation. The WSAMI environment supports such a feature by introducing the situation-sensitive composition of any requested Web service, which is realized through the discovery of service instances that serve matching the service's specification in terms of both functional *and* non-functional properties, as outlined below.

3.2 Enriching the Web Services Architecture for Ambient Intelligence

Situation-sensitive composition provides the basis for enabling the development of ambient intelligence applications: requested services are composed according to service instances that may be retrieved in the environment, including middleware-related ones for the sake of quality of service. The selection of service instances that are integrated relies on the provided specification. Since the process takes place in an open environment, security issues arise. We adopt an approach based on certificates for dealing with security-related requirements. Nodes hosting and accessing services obtain certificates from relevant certificate authorities, and certificates are used for authentication between interacting nodes. A related issue is the one of the correctness of service specification with respect to the service behavior, which is an area for future work. In general, off-line analysis of Web services with respect to provided specification needs to be addressed for the sake of robustness. Certification of services with respect to specification may then be exploited for keeping low the run-time overhead associated with situation-sensitive composition, while enforcing robustness. This paper concentrates on the definition of the core of the WSAMI environment enabling situation-sensitive composition. The behavioral specification of Web services using WSAMI for supporting off-line analysis and enforcing robust composition of services is beyond the scope of this paper. Note that results from the software architecture community, e.g., [Medvidovic and Taylor, 2000, Issarny et al., 2002], but also ongoing work in the area of Web services, e.g., [Narayanan and McIlraith, 2002], already provide us with sound base ground to address this issue.

Effective situation-sensitive composition of Web services is realized in the WSAMI environment by supporting: (i) a composition process that is distributed over the nodes hosting the component services, (ii) connectors customization to enforce quality of service, and (iii) exploitation of today's WLAN for both local- and wide-area service discovery.

Distributed composition of services. Assisting the composition of Web services in a stationary environment has been an active area of research over the last couple of years, e.g.,

see [Tartanoglu et al., 2003] for a survey. Proposed solutions relate to offering an XML language for describing the overall composition process (also called Web services orchestration), which may further be complemented with an XML language for specifying conversations (also called Web services choreography), so as to ensure correctness of the interaction protocols with individual services. The resulting description primarily serves as a composition design, since the implementation of the associated composite service relies on development using specific platforms. A composite service may then adhere to different execution patterns such as central coordination and peer-to-peer execution [Benatallah et al., 2002]. Dynamic integration of Web services is further supported by using UDDI that defines a middleware-related trading service for retrieving service instances that match a given service specification (e.g., WSDL specification). The above provides adequate solutions towards developing composite services whose component services may possibly be retrieved dynamically. However, the dynamic integration of services shall be embedded within the implementation of individual services and is left under the responsibility of the service's developer.

Dynamic selection and integration of Web services shall be automated in the ambient intelligence context. It must thus be supported by the underlying middleware infrastructure, as opposed to requiring specific handling by the developer. The WSAMI language enriches the specification of Web services with the abstract specification of services with which interaction is required. The WSAMI middleware further supports retrieval of matching service instances in the environment. Selection of the service instances making up the overall configuration of a given composite service may be either handled in a central location as in Cans (e.g., the node initiating the request to the top-level composite service) or decentralized (e.g., distributed over the nodes that run composed services that invoke other services). The latter solution is the one implemented in WSAMI, as it must be preferred in the mobile setting. This is not solely due to the resulting load balancing, but also to the connectivity of the various nodes, as the services accessible from one node may not be from another.

Enforcing quality of service via connector customization. Quality of service is a primary concern in the mobile domain due to inherent variability in available resources, and the need for minimizing energy consumption and enforcing security in the wireless setting. Minimizing resource consumption and in particular energy consumption shall be accounted for in the design and implementation of the base middleware infrastructure. In particular, core middleware functionalities must introduce minimal computation and communication costs, the latter inducing significant energy consumption. Enforcement of quality of service shall further be addressed according to the application's specifics, i.e., the application-related Web services that are composed. A well-known solution to this issue relates to connector customization by integrating the necessary middleware-related services [Issarny et al., 2002]. It has in particular be applied to the mobile domain by the Cans infrastructure [Fu et al., 2001]. However, Cans handles connector customization in a central location, which is not suitable in general. As for the selection and integration of application-related Web services, the ones of middleware-related Web services must be distributed over participating nodes, each node dealing with the customization of the connectors supporting the interactions with the services it invokes. Customization may then be either explicitly addressed by the developer through the specification of an associated composite service as for application-related services, or handled by the environ-

ment given the abstract specification of required non-functional properties. The latter solution is the one implemented in WSAMI, which offers significant advantages, including: enforcement of separation of concerns, and greater flexibility in used services.

Our solution to the systematic customization of connectors lies in the WSAMI support for: (i) abstractly specifying non-functional properties to be enforced over connectors; and (ii) customizers that define (middleware-related) Web services to be integrated for enforcing a given non-functional property. The specification of non-functional properties should be agreed upon by all Web services. Although there does not yet exist such a standard, it may be anticipated that related standards will become available, considering in particular ongoing work on the specification of Service Level Agreements (SLA) for Web services, e.g., see [Sahai et al., 2002]. SLA specification in particular targets the precise description of non-functional properties (performance, quality, etc.) offered by services (e.g., see [Lamanna et al., 2003]), and as such is of direct relevance to our work. At this stage, we use simple XML documents for the specification of non-functional properties, which suffice to assess our customization approach, while it is part of our future work to use relevant standard schemas as they become available.

Combining local and wide-area service composition. Ubiquitous networking relies on exploiting the various networking technologies that are now available. In particular, the WLAN technology appears as a key enabler of ubiquitous networking, as it is replacing wired LANs [Ritter, 2003], and is being extended to provide, e.g., greater bandwidth, quality of service, and enhanced security. Today's WLAN supports both direct interaction among mobile nodes that are in the communication range of each other (ad hoc mode) and interaction via a base station (infrastructure-based mode). In addition, ad hoc routing protocols enable interaction between mobile nodes that are not in the communication range of each other without requiring any infrastructure [Poor et al., 2003], if a path of mobile nodes may be found to route messages. Thus, given the WLAN technology, services may be accessed in the local and/or wide-area (i.e., Internet) depending on the service instances that are available over the local network and/or whether a base station is reachable. The WSAMI core middleware allows for service composition both in the local- and wide-area over WLAN, via its naming and discovery service, which is the only middleware-related service that we introduce in addition to the SOAP-based core broker of the Web services architecture.

4 WSAMI Language

The WSAMI language enables the specification of Web services so that they can be dynamically composed according to the environment in which services are requested, while enforcing quality of service. The WSAMI specification of a service relates to the service's abstract interface (§ 4.1), possibly embedding associated non-functional properties (§ 4.2). The following introduces the XML schema defining the WSAMI language, through specification samples associated with the collaborative schedule service that was discussed in Section 3.1; the interested reader is referred to the appendix for the complete WSAMI XML schema.

```
<Abstract name=‘‘CollaborativeSchedule’’>
  <Interface hrefSchema=‘‘http://example.com/schedule/ScheduleRequest.wsdl’’/>
  <Conversation hrefSchema=‘‘http://example.com/schedule/ScheduleInteraction.cdl’’/>
</Abstract>
```

a. Abstract interface

```
<Service name=‘‘CollaborativeSportSchedule’’>
  <Abstract hrefSchema=‘‘http://example.com/schedule/CollaborativeSchedule.wsami’’/>
  <Concrete hrefSchema=‘‘http://example.com/schedule/ConcreteSchedule.wsdl’’/>
  <Required>
    <ReqService name=‘‘Agenda’’ instance=‘‘true’’ multiple=‘‘true’’>
      <Abstract hrefSchema=‘‘http://example.com/schedule/AgendaSchedule.wsami’’/>
    </ReqService>
    <ReqService name=‘‘MeetingPlace’’ instance=‘‘true’’>
      <Abstract hrefSchema=‘‘http://example.com/schedule/MeetingPlace.wsami’’/>
    </ReqService>
  </Required>
</Service>
```

b. Instance interface

Figure 1: WSAMI specification of service interfaces

4.1 Abstract Interfaces and Instances

Using WSDL [W3C, 2002], a Web service specification embeds: (i) the service's abstract interface that describes the messages exchanged with the service, and (ii) the concrete binding information that contains specific protocol-dependent details including the network endpoint address of the service. In our context, the first part is the basis for locating services, while the latter is associated with service instances that are dynamically retrieved. Retrieval of a service is then based on the matching of the abstract interface associated with the requested service, with the abstract interfaces of reachable instances. However, interaction with a service should also comply with the protocol that is assumed, i.e., the conversation (also termed choreography) that must be realized. Such a specification is being addressed by the Web Services Choreography Working Group. Hence, as far as the specification of the service's functional behavior is concerned, the definition of the related WSAMI `Abstract` element is direct from existing Web services languages, as illustrated by the abstract interface associated with the collaborative schedule service (see Figure 1.*a*).

The WSAMI `Service` element that is associated with any service instance specifies the WSAMI abstract interface of the service together with concrete binding information. In addition, the execution of a service on a node may be dependent upon the execution of other services, which must be retrieved in the environment. WSAMI allows specifying the required services that are mandatory, which will be checked for availability prior to running the embedding service. Required services that are not specified will be located upon actual access, possibly leading to raise an exception whose handling is application-dependent. Still using

the collaborative schedule service for illustration, Figure 1.*b* gives the WSAMI specification associated with a service instance. In the specification, the attribute `instance` associated with required services serves stating that the request for the embedding service will provide the URI of the instance, while the `multiple` attribute serves stating that multiple instances of the service get composed.

Situation-sensitive composition relies on a matching relationship defined over WSAMI abstract interfaces. Given the request for a service identified by its WSAMI abstract interface, a service instance whose abstract interface (i.e., value of the `Abstract` element) *matches* the one of the requested service, is sought in the environment. Two abstract interfaces are then said to match if their respective documents are syntactically equal. This allows us to keep to a minimum the processing cost associated with checking specification matching: two abstract interfaces match if the related WSAMI documents have the same URI.

The above matching relationship further suggests the development of Web services for ambient intelligence through the reuse of declarative specifications of services that are made available over the Web. This is consistent with the approach that is put forward for the development of Web services in general through the provision of *universal registries* as, e.g., enabled by UDDI.

4.2 Enforcing Non-functional Properties

A key feature of WSAMI lies in the specification of non-functional properties associated with services, so as to enforce non-functional properties in terms of at least security and performance, i.e., the two mandatory criteria for the consumer acceptance of ambient intelligence systems, independent of the users and service providers.

Specifying non-functional requirements. Non-functional requirements decompose into the non-functional properties that must be provided by the service itself (i.e., built in the service implementation) and the ones that must be enforced at the connector level. The latter specifies customization of the middleware for interaction with the service, i.e., middleware-related services that need to be integrated on the network path between the interacting services.

The WSAMI specification of abstract interfaces then extends with the definition of the `ServiceQoS` and `ConnectorQoS` elements that set the non-functional properties associated with the service and connector, respectively. Properties are currently simply defined by string values. However, the specification of non-functional properties may be extended so as to support off-line analysis of Web services' non-functional behavior, and will later be revised according to standard languages emerging for quality of service specification (e.g., languages for service-level agreements). For example, consider the collaborative sport schedule service, the enriched specification for the abstract interface associated with the agenda service is given in Figure 2.*a*, and related definition of non-functional properties is given in Figure 2.*b*.

As defined in the previous section, a service instance matches a requested service if they both specify the same document URI for the service's abstract interface. Hence, this enforces matching behavior of services with respect to *both* functional properties and non-functional properties stated in the abstract interface. Increased flexibility could be obtained by not strongly coupling the specification of non-functional properties with abstract interfaces. However, our solution

```
<Abstract name=‘‘AgendaSchedule’’
      xmlns:nf=‘‘http://www-rocq.inria.fr/arles/wsami/’’>
  <Interface hrefSchema=‘‘http://example.com/schedule/AgendaScheduleRequest.wsdl’’/>
  <Conversation hrefSchema=‘‘http://example.com/schedule/AgendaScheduleInteraction.cdl’’/>
  <ServiceQoS>
      <QoSCriterion name=‘‘nf:transactionalService’’/>
      <QoSCriterion name=‘‘nf:security’’/>
  </ServiceQoS>
  <ConnectorQoS>
      <QoSCriterion name=‘‘nf:security’’/>
  </ConnectorQoS>
</Abstract>
```

a. Specifying non-functional requirements

```
<xsd:simpleType name=‘‘QoSType’’>
  <xsd:restriction base="xsd:string’’>
    <xsd:enumeration value=‘‘transactionalService’’/>
    <xsd:enumeration value=‘‘security’’/>
    <xsd:enumeration value=‘‘saveBandwidth’’/>
  </xsd:restriction>
</xsd:simpleType>
```

b. Definition of non-functional properties

```
<Customizer name=‘‘EnforceSecurity’’
      xmlns:nf=‘‘http://www-rocq.inria.fr/arles/wsami/’’>
  <QoSCriterion name=‘‘nf:security’’/>
  <Local hrefSchema=‘‘http://example.com/NF/SymCrypto.wsami’’/>
  <Remote hrefSchema=‘‘http://example.com/NF/SymCrypto.wsami’’/>
</Customizer>
```

c. Customizer for security

Figure 2: WSAMI specification of non-functional properties

allows for an efficient retrieval and customization process, and further minimizes associated resource consumption that is crucial for wireless, resource-constrained devices.

Automated connector customization. If non-functional properties must be enforced over the connector, as specified using the `ConnectorQoS` element, the connector must be customized. Connector customization may lead to quite complex interactions among services and cannot be realized automatically in general [Kloukinas and Issarny, 2000]. However, such automated customization can be supported for non-functional properties that are enforced through middleware-related services that adhere to the pipe&filter architectural style by *filtering* the content that is sent over the network. The handling of non-functional properties that do not meet the above requirement should thus be dealt with by the Web service developers who should either make explicit the related Web service composition, e.g., to enforce dependability [Tartanoglu et al., 2003], or use a proprietary middleware for content delivery (e.g., delivery of

continuous content) where WSAMI may still be exploited for initiating the service in the latter case (e.g., negotiating quality of service).

Content filtering may be achieved using specific proxy nodes, e.g., for mobile terminals [Fox et al., 1998], possibly leading to introduce a new system of protocols and document types for interaction with the wireless client as in the WAP (Wireless Access Protocol). An alternative approach is to use a pair of content customizers at both ends, as presented in [Steinberg and Pasquale, 2002]. Our approach is based on the latter, which may be exploited for dealing with security and performance requirements but also other non-functional properties such as reliability as presented in the aforementioned reference. For instance, security-related customizers allow for encrypting sensitive information that passes over insecure HTTP connections [Steinberg and Pasquale, 2002].

A *customizer* decomposes into a local and a remote service, i.e., the former must be available in the environment of the client while the latter must be in the environment of the server. Then, any message exchanged between the client and the server goes through the customizer. For illustration, Figure 2.*c* gives the WSAMI specification of a customizer enforcing security through the use of symmetric cryptography for the encryption of messages: the `QoSCriterion` part specifies the specific non-functional property that is enforced by the embedding customizer, and the `Local` and `Remote` parts specify the abstract interfaces of the customizer services, which must be run *close* to the client, and to the server, respectively. In our context, a customizer service is considered close to its associated end-point service if it can be reached from the service's host, but security-related customizer services that are required to be run on the same host as their associated end-point. Customizers that require co-location with associated end-point are identified by extending the `Customizer` element with `<xsd:attribute name=‘‘Colocation’’ type=‘‘xsd:boolean’’ default= ‘‘false’’/>`.

Multiple non-functional properties may be required for interaction with a single service, which raises the issue of composition of middleware-related services. Services are composed according to the ordering of requirements for associated non-functional properties in the service specification. More specifically, services of the customizers are chained according to the ordering of requirements specification. The only exception is for the security customizer that is always composed at the end.

The benefit of using customizers is dependent upon the environment in which the service is requested. While enforcing security is almost always mandatory, enhancing performance through filtering is dependent upon available network bandwidth and connectivity. We rely here on the specific implementation of customizers to adapt to resources availability. A more flexible solution could be undertaken by allowing to specify constraints over resources availability for both the use of customizers and non-functional requirements. However, this would lead to more complex computation upon composition and also require more cooperation from the local environment, which contradicts our goal of introducing a minimal core middleware.

Figure 3 summarizes specifics of the WSAMI languages with respect to the elements of the Web services architectures: the description of Web services is enriched with the specification of required services and non-functional properties to allow for the situation-sensitive composition of services, as enabled by the WSAMI middleware detailed in the next section. Non-functional requirements are further exploited to customize connectors accordingly, using customizers that are dynamically integrated.

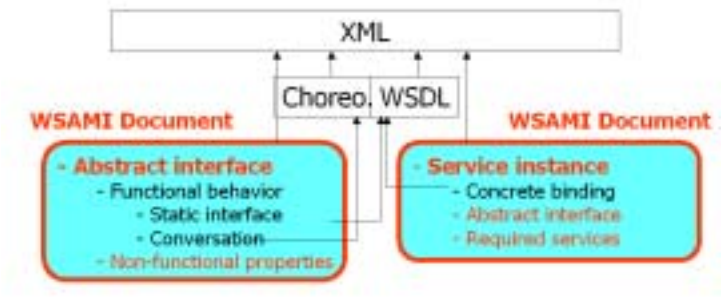

Figure 3: WSAMI description of services

5 WSAMI Middleware

The core middleware associated with Web services lies in the provision of a SOAP-based core broker, including SOAP containers that are able to deploy Web services and to manage RPCs from SOAP clients and dispatch them to services. In our context, Web services may be deployed on possibly resource-constrained, wireless terminals. However, we do not consider that the deployment of Web services on mobile platforms is a major issue given base run-time platforms being developed for such terminals (e.g., see Connected Device Configuration of Java2 Micro Edition at http://java.sun.com/products/cdc). We have in particular implemented a Java-based core broker prototype for Web services, which can be deployed on PDA-like devices, as presented in Section 6.

This section concentrates more specifically on the design of the WSAMI naming&discovery middleware service (simply referred to as ND service in the following), which supports the situation-sensitive composition of services, given the services' WSAMI specification. We are in particular interested in exploiting both ad hoc and infrastructure-based modes of WLAN for enhanced connectivity and hence enhanced service availability. The ND support for dynamically locating requested services lies in: the management of repositories of services' abstract interfaces and instances (§ 5.1), and locating instances of services that are reachable both in the local and in the wide area (§ 5.2). In addition, the ND service handles connector customization together with authentication of service instances (§ 5.3).

5.1 Service Repositories

The ND service manages two repositories, i.e., the *local repository* and the *remote repository*. The former stores information about services instances that are locally supported; the latter stores information about known remote instances.

The remote repository acts as a local cache whose content evolves according to the history of user requests. The size of the repository is in particular set according to available storage, and entries are removed according the cache's replacement policy, which adheres to the Least Recently Used (LRU) policy. Note that advanced automatic prefetching techniques with respect to the user's profile might be exploited, although not part of our current design. Our concern of minimizing resource consumption on mobile terminals together with the fact that

our approach relies on *universal repositories* of Web services specification (including WSAMI specifications) lead us to not store nor process XML documents on the terminals for service discovery. Instead, only related URIs are exploited, which is enabled by the specific design choices for WSAMI, and in particular the definition of the WSAMI matching relationship. Considering the associated development of ambient intelligence applications, developers will exploit the WSAMI services specifications from the universal repositories, allowing to generate the client proxies for interaction with service instances whose abstract interfaces match the target service. Each element of the *remote repository* stores necessary data for establishing interaction with known remote service instances matching a given abstract interface, i.e., each element includes:

- The URI of the WSAMI document defining a service abstract interface.
- The list of *known* matching service instances that decomposes into: (i) the list of known instances available on the Internet, and (ii) the list of instances discovered in the local environment, through the ND service support for their discovery as discussed in the next section. Each element of the above lists includes: (a) the URI of the corresponding service, (b) the actual binding information, and (c) information about related customizers-if any, and (d) for instances available in the local area, a tag that specifies whether the hosting node is power-plugged or not.

Upon a service request to the ND service that provides as input the URI of the document characterizing the service's abstract interface, the corresponding entry is sought in the repository. The entry may be missing, in which case one is created and a matching instance is sought in the environment (unless already specified in the request), using the ND support for service discovery, which relies on the local repository.

Each element of the *local repository* stores information characterizing a service instance that is available locally, i.e., each element includes:

- The URI of the document defining the corresponding abstract interface (i.e., `Abstract`).
- The URI of the document defining the corresponding service (i.e., `Service`).
- The URI of the WSDL document defining the corresponding concrete interface (i.e., `Concrete`).
- The list of required services -if any-, which provides the URI of the document defining the abstract interface of each required service (i.e., `Required`).
- The list of URIs of the documents defining the abstract interfaces of the customizer services that need to be integrated on the communication path with the given service, according to the non-functional properties that need to be enforced over the connector (i.e., `ConnectorQoS` part of `Abstract`).

Note that the last item follows from selecting (abstract) customizers at the time the service is deployed on the terminal, as opposed to retrieving on-line, the customizers' abstract specifications from specified non-functional properties. In addition, for customizer services that need

to be co-located with their associated end-point (i.e. the attribute `Colocated` is set to `true`), instances of the remote customizers are locally deployed together with their end-point. This still follows from our concern of minimizing computation and communication costs associated with situation-sensitive composition, for effective realization over wireless nodes.

Finally, we consider that any device possibly hosts a Web services directory (e.g., personal directory) that may be browsed or directly queried for retrieving a specific service instance known to the user. It further stores the URI of the user's favorite universal repository (i.e., UDDI-like repository).

5.2 Locating Service Instances

The above repositories provide the base functionalities for supporting the following operations:

- Request for a service given the URI of an abstract interface.
- Request for a specific service instance.

Consider first the handling of a request for a service given its abstract interface, since the handling of requests for specific instances is direct from it. A local instance is first searched locally through a request to the local repository. If an instance is available, request for the service's execution may proceed. Otherwise, a remote instance needs to be retrieved. Nodes that are contacted then depend on the underlying wireless network, including whether it is infrastructure-based or ad hoc. Using a WLAN such as IEEE 802.11b, the network may be run in either the ad hoc or infrastructure-based mode, where the latter requires availability of a base station in the local communication range. We further consider that the network may be switched from one mode to another (see next section for specific prototype implementation), depending on the availability of a base station and whether the user accepts to be charged for communication (if applicable). The ad hoc mode of WLAN in particular offers a number of advantages, including: communication does not incur any financial cost for the consumer, and improved connectivity is achieved by not requiring the presence of a base station in the communication range. Then, instances may be retrieved either via the Internet if the network is in the infrastructure-based mode or via nodes in the local communication range in either mode. The retrieval of instances that are available in the local environment relies on an underlying service location protocol [Bettstetter and Renner, 2000]. Using such a support, nodes running the ND service may be discovered in the local environment, leading to get necessary binding information as well as to register whether the nodes are power-plugged or not. Given the request for a service that does not specify a specific instance, a matching instance is retrieved according to the following process.

- If the network is in the infrastructure-based mode, the following sequential steps are performed until an instance is found:

  I1. The service instance is sought in the local area by first looking for eligible instances hosted by power-plugged nodes within the remote repository. If there is no such reachable instance, a request for a service instance is sent to the ND services in the

local area that are run on power-plugged nodes. Upon reception of the request, the remote ND service seeks matching instances within the local repository and returns the eligible instances -if any.

I2. The service instance is sought on the Internet by looking for eligible instances within the remote repository and local Web services directory -if any.

I3. A reachable service instance hosted by a non power-plugged node in the local area is sought within the remote repository. If none is found, the service is requested to the ND services run on the (non-power-plugged) wireless nodes in the local area.

I4. Ultimately, the service request is sent *via* the Internet, to the universal repository that is provided by the user upon ND initialization.

- If the network is in the ad hoc mode, the service is requested to nodes in the local area, according to the following steps, until an instance is found:

  A1. A reachable service instance hosted by a power-plugged node is sought in a way similar to Step I1.

  A2. The same process as above is applied for non power-plugged nodes in a way similar to Step I3.

  A3. An instance of a service that relays messages for access to the Internet is sought, according to the above Steps A1 and A2. If an instance is found, the target service is sought within the Internet according to the above Steps I2 and I4.

  Note that increased availability may be achieved by exploiting ad hoc routing protocols as, e.g., addressed in [Sailhan and Issarny, 2003] for the specific case of content caching, which we are extending to service caching for further enhancement of the WSAMI middleware.

In the case where no instance is retrieved, an exception is signaled to the requesting service. If one or more eligible instances are retrieved, the remote repository is updated accordingly. And, in the case where multiple instances are retrieved, one is chosen randomly, although Quality of Service (QoS) could still be improved using QoS-aware service location, which we are currently investigating [Liu and Issarny, 2004]. The handling of a request for multiple instances of a given service is straightforward to infer and thus not detailed.

The handling of a request for a specific service instance follows from the above presentation; the instance is sought in the environment according to the networking environment. Finally, note that in the case of local service instances that explicitly declare required services, related instances are sought in the environment according to the above selection process, prior to the service execution.

5.3 Connector Customization

The above retrieval process must be enriched to cope with non-functional requirements, which decompose into: (i) authenticating the nodes with which interactions take place so as to enforce communication with trusted nodes, and (ii) customizing connectors.

The service instances that are selected need to be authenticated for the sake of security. In our context, authentication subdivides into: (i) authenticating an instance with respect to its published interface (i.e., the instance does realize the abstract interface it advertises), (ii) authenticating a given instance. The former applies to instances that are selected at runtime given an abstract interface and thus relates to authenticating the related ND services, the latter applies to authenticating instances that are specified in the service request. We rely on certificates for both types of authentication, i.e., certificates associated with services are obtained from certificate authorities by clients and service providers. The certificate authorities that are accessed as well as the period of validity of the certificates depend on the specifics of the services, including whether it is public or private, and the level of security that is required. In the worst case, the certificate on the terminal may be no longer valid since its connectivity enabled interaction with the certificate authority. In this case, authentication may not be possible, leading to notify temporary service unavailability to the user.

In the case where a remote instance is accessed, connector customization is necessary if non-functional requirements are stated in the `ConnectorQoS` element of the service's abstract interface. The WSAMI middleware running on the client then seeks instances of the local and remote customizers, as provided by the information associated with the specific instance when retrieved. The location process is the same as the one presented in the previous section. However, note that in the case of customizers requiring co-location (i.e., the `Colocation` attribute is set to `true`), including in particular security-related customizers, the local and remote customizers must be located on the nodes running the client and the service, respectively. Once instances of the customizers services are retrieved, the middleware sets up the interaction chain to allow each message to follow the path from the client to the service via the chain of local and remote customizers, instead of the simple client-service interaction. Service interaction via customizers is transparent to the client, which uses the same stub to call either the service or the local customizer. The middleware takes in charge for every SOAP message exchanged via a customized connector, to add the required information to allow the interaction chain to be set up between client, customizers and service.

6 Prototype Implementation and Evaluation

In order to assess our solution, we have developed a first Java-based prototype of the WSAMI core middleware, which has been used to implement the demonstrator realizing the scenarios discussed in Section 3.1 (see Figure 4). We use IEEE 802.11b as the underlying WLAN. All the services involved in the demonstrator are implemented as Web services on top of WSAMI, which allows for the deployment and the dynamic discovery and composition of Web services in the mobile environment. In addition to services that are specific to the application, the demonstrator integrates services enabling intelligent aware interfaces through multi-modal user interaction combining speech and gesture [OZONE, 2003]. Complexity associated with the implementation of the demonstrator-specific services relates to their functionalities, while WSAMI relieves the developers from dealing with mobility management. In addition, as detailed in this section, WSAMI allows for service access and deployment on resource-constrained devices with performance comparable to existing middleware for the wired environment.

Figure 4: Seamless access to services in the mobile environment using WSAMI

The WSAMI core middleware prototype subdivides into: (i) the WSAMI SOAP-based core broker (§ 6.1), including the CSOAP SOAP container for wireless, resource-constrained devices (§ 6.2), (ii) the Naming&Discovery (ND) service (§ 6.3), including support for connector customization (§ 6.4). Figure 5 depicts the main components of the WSAMI prototype implementation on top of which Web services execute; greyed components denote available implementation that we reuse while the components that we developed are highlighted in bold face. Note that components developed as part of the WSAMI core broker belong to any Web services platform; a new implementation has been provided so as to allow for execution on resource-constrained devices.

6.1 WSAMI Core Broker

As depicted in Figure 5, the WSAMI core broker subdivides into:

- An XML parser for which we use the Xerces implementation[2].
- A SOAP container that is able to deploy Web services, and to manage RPCs (Remote Procedure Call) from SOAP clients and dispatch them to services. The SOAP RPC specification requires a container that is able to receive RPC requests and send RPC replies using the HTTP protocol for communication between clients and servers. SOAP messages are coded in XML. Thus, every container is closely coupled with an XML parser to translate XML messages into local operation calls, as supported by the container's runtime environment, and conversely. SOAP containers are already available for a number of platforms. However, as discussed in the next section, we had to develop a SOAP container, called CSOAP, that can be accommodated by resource-constrained nodes. The SOAP container includes a compiler for generating Java stubs, from the WSDL interface definition, e.g., the WSDL2Java compiler of Axis (see http://ws.apache.org/axis/). Note that the compiler is part of the development support and thus not integrated within core brokers deployed on end-users terminals.

[2]see http://xml.apache.org/xerces2-j/

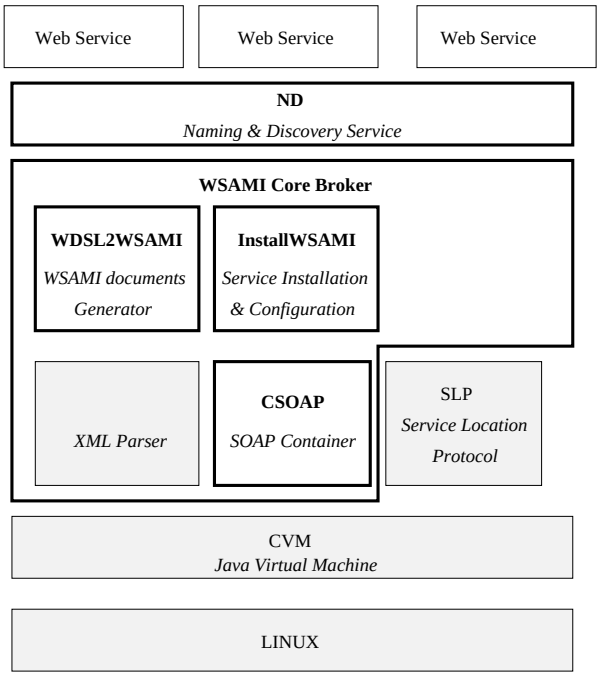

Figure 5: WSAMI core middleware prototype

- The WSDL2WSAMI tool that generates the WSAMI documents that describe WSAMI-enabled Web services (i.e., abstract interface, service definition, customizer-related specification). The inputs for this utility are the related WSDL files, and the WSAMI-specific definitions relating to quality of service and required services. As for the compiler embedded in SOAP containers, the tool is part of the development support and is thus not deployed on end-user terminals. Note that our current prototype does not deal with the choreography specification associated with services, which is in particular due to the fact that related language standard is being under definition.

- The InstallWSAMI tool to install and configure services on the WSAMI middleware. This utility serves installing all the WSAMI, WSDL and implementation files that define Web services, on the middleware, and thus making available the services for calls from clients. The tool is implemented as a client sending information over the HTTP protocol and a server-side running on a SOAP container supporting servlets.

From the developer's standpoint, the development of WSAMI-enabled Web services adds minimal overhead compared to the one of base Web services. The developer has to specify services and non-functional properties required by the service being developed using the WSAMI language.

6.2 The CSOAP Container for Wireless, Resource-constrained Terminals

Running Web Services on a wireless system integrating a Java-based platform requires a version of the Java Virtual Machine (JVM) that targets devices with limited resources and a lightweight SOAP container, but still providing essential tools to deploy services into the container. Regarding the availability of JVMs for resource-constrained devices, the Java Community Process has defined a configuration of the Java 2 Platform, Micro Edition (J2ME) that is the Connected Device Configuration (CDC). CDC includes the CVM virtual machine (Java VM for Consumer electronics), a full J2SE-compliant virtual machine to support Java applications on consumer electronic and embedded devices such as smart communicators, pagers, PDAs, and interactive, digital television set-top boxes. Typically, these devices run a 32-bit microprocessor/controller and have more than 2MB of total memory for the storage of the virtual machine and libraries. Not every class from each J2SE package is supported. However, those classes that are supported behave identically and have the same public interfaces as those of J2SE. There are also profiles being developed to extend the features offered by CDC to fill the gap with J2SE. The first profile, the Foundation Profile, is released with CDC and generally, the two are meant to go hand-in-hand. The Foundation Profile extends CDC by adding most of the missing J2SE core libraries, except for those related to java.beans, java.rmi and java.sql packages. In particular, CDC and the Foundation Profile (FP) do not include the RMI (Remote Method Invocation) package that is required from available SOAP implementations to execute remote procedure calls. Reference implementations of CDC and the Foundation Profile are available for Linux and VxWorks platforms.

To cope with the limitations imposed by CVM and resource-constrained devices, we have developed the CSOAP SOAP container prototype, specifically targeted to these platforms. The prototype implementation that we have developed is mainly based on Sun's JAX-RPC (Java API for XML-based RPC) specification (see http://java.sun.com/xml/jaxrpc/) that is a specification aimed at enabling the development of SOAP-based interoperable and portable Web services. JAX-RPC provides the following core API for developing and deploying Web services on the Java platform:

- The interfaces for supporting the dynamic invocation of a service endpoint and the definition of the standard interface for the stub used as proxy to make calls to target service instances.
- The interfaces to manage the marshalling/unmarshalling of data types.
- The interfaces for modifying the default components chain that is responsible for handling a SOAP message both on client and server sides, and both in request and response phases. The components added into the chain have the control (read/write access) over every SOAP message.

Together with the CSOAP container, we have developed the InstallWSAMI tool that provides deployment and configuration functionalities. The utility for deploying and undeploying Web services is based on a simple XML-based language, specifically defined for the purpose of minimizing the complexity of the system. A related tool for the automated generation of stub

files is also offered to allow an easy development of Web Services on the server side and of the corresponding client applications. The input data required is the WSDL file that defines the abstract interface of the service and the output consists of the Java files that will be used to develop a service that will run on the SOAP container and consequently will be based on Java classes and interfaces included in the SOAP server implementation. The core WSAMI middleware and related tools described in the previous section can be also applied to our CSOAP container implementation for CVM and resource-constrained devices.

The memory footprint of our CSOAP implementation is of 90KB, as opposed to the 1100KB of the Sun reference implementation. The overall memory footprint of our Web services platform is of 3.9MB, dividing into 3MB for the CVM, and 815KB for the Xerces XML parser in addition to the CSOAP implementation.

We did a number of experiments to investigate the performance of our lightweight Web services platform, and in particular assess it against an existing Web services platform but also against traditional middleware platforms being considered for the mobile environment. We concentrated only on the performance of the core broker, since the performance associated with application-specific operations derives from the one of the CVM. We implemented a service that offers an operation taking as input an array of integers (`int` elements) and returning a string upon reception of the message. The response time of the above operation call then subdivides into the time taken for the marshalling/unmarshalling of the array for issuing the SOAP request and the time of message exchanges. Hence, this allows characterizing the impact of processing XML messages.

We experimented with both PDAs and laptops: (i) the PDAs are Compaq iPAQ 3600 with a 206MHz Strong-ARM CPU and 32MB RAM running the Linux Familiar 0.6 distribution and the J2ME CDC 1.0.1 JVM, (ii) the laptops are Compaq PCs with a 500MHz Intel PIII CPU and 192MB RAM running the Linux Redhat 7.3 distribution and the J2ME CDC 1.0.1 JVM with RMI optional package. The wireless LAN is IEEE 802.11b (Lucent 11MB WaveLAN PC Card).

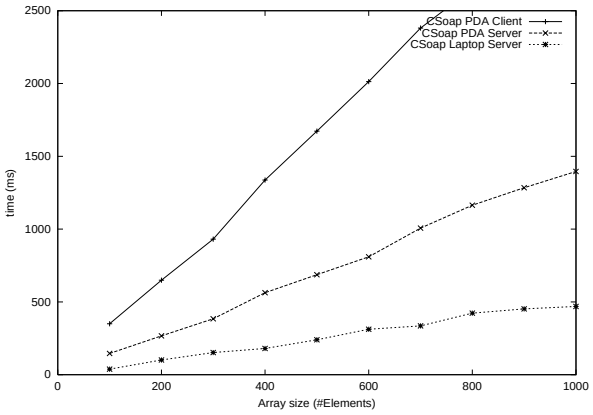

Figure 6: CSOAP response time for large arrays

Figures 6 and 7 give the response time of the operation call on the client and of the call processing on the server, according to the size of the array (i.e., XML parsing complexity)

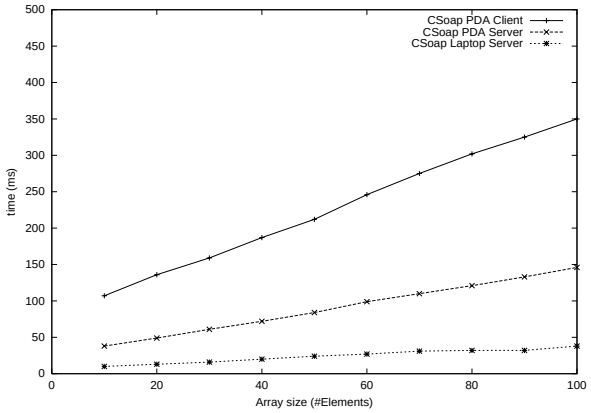

Figure 7: CSOAP response time for small arrays

when both the client and the server run on a PDA and interact via the WLAN in the ad hoc mode, for arrays of up to 1000 elements (Figure 6). It further gives the request processing time on the server when the server runs on the laptop for characterizing the performance impact of the PDA. Further analysis of the performance shows that 80% of the time taken by the SOAP container for call processing on the server is due to the XML parser for unmarshalling the message. Note that the most relevant figures are the ones obtained with arrays of up to 100 elements (Figure 7) since this corresponds to handling operations with up to 100 parameters of simple type. Results further show that the response time for the PDA client is about three times the processing time on the PDA server, and hence that message exchange takes about the same time as XML message processing in the context of an 11MB WLAN.

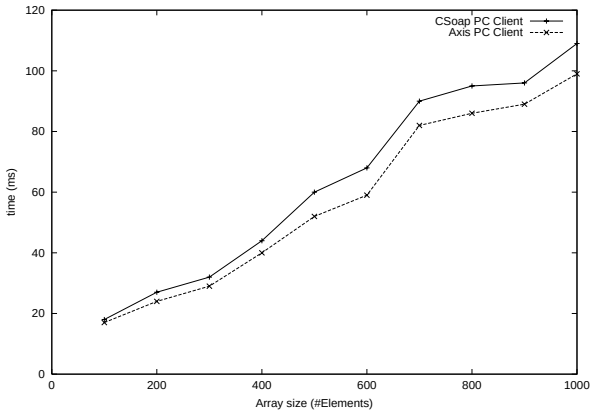

Figure 8: CSOAP versus Axis

Figure 8 compares the performance of the CSOAP server with Axis when both execute on a PC with 800MHz PIII CPU and 384MB RAM running Linux Redhat 7.3 distribution and the J2SE 1.3.1 JVM. It gives the response time of the call processing on the server according to the size of the array. Results show that our prototype implementation is about 10% slower

than Axis. We expect to obtain similar performance with the next version of our prototype, by tuning further the implementation. However, the major source of performance improvement for CSOAP, and for SOAP containers in general, lies in providing a more efficient XML parser.

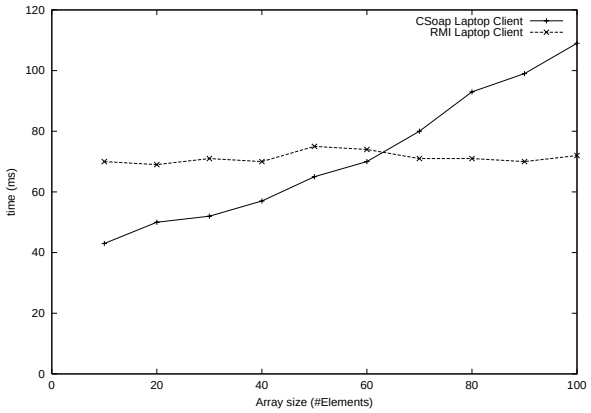

Figure 9: CSOAP versus Java RMI

Finally, Figure 9 compares the performance of CSOAP with the one of Java RMI, which is among traditional middleware considered for distributed mobile computing. The client and the server run on a laptop and interact via the WLAN in the ad hoc mode. The figure more specifically gives the response time of the operation call on the client when the server laptop runs either a CSOAP container, or an RMI server using the RMI optional package combined with the Sun RMI registry implementation for Intel platform (given associated modification of the client and service code used for experiment). Results show that the CSOAP-based interactions offer better performance for an array of up to 60 elements, while Java RMI offers better performance for larger arrays. Since it is expected that most operation calls have less than 60 parameters for the target ambient intelligence application domain, it can be concluded that the CSOAP middleware compares favorably with more traditional middleware platforms for mobile computing. This further shows that offering a solution based on Web services for ambient intelligence is not only beneficial by supporting pervasive services but is also effective from the standpoint of performance.

6.3 Naming and Discovery Service

The ND service that complements the above core middleware is implemented according to the service design introduced in Section 5.2, where we use the OpenSLP Java implementation (see http://www.openslp.com/) of the SLP (Service Location Protocol) standard as follows:

- On every host, an SLP server runs under the Service Agent (SA) mode
- The ND service registers itself on the SLP server. The latter has the responsibility to multicast the address of the local ND service to all the reachable ND services on the network.

- The ND service collects all the ND service addresses from the SLP server. When it receives a lookup request for a service, it contacts all the (authenticated) ND services in its list to obtain all the service instance addresses available on the local network.

We further have implemented a service enabling interaction among mobile nodes, independent of the networking mode they are using (i.e., ad hoc or infrastructure-based). If a mobile terminal is in the ad hoc mode and switches to the infrastructure-based one, the packets that are sent to this mobile terminal are lost (and conversely). The cost associated with the use of two wireless cards can be prohibitive, and prevents from having a connection with the infrastructure and the ad hoc network at the same time. Thus, to allow access to the Internet whereas a mobile terminal is in the ad hoc mode, we have implemented a service that provides access to the Internet to mobile terminals that are in the ad hoc mode. Such a service may be offered by any terminal supporting two wireless interfaces, which can be a power-plugged, resource-rich mobile terminal or a base station. We use the NAT (Network Address Translation) protocol (see http://www.ietf.org/rfc.html, RFC 3022) so as to allow linking the ad hoc network with the outside network. This protocol modifies the network address contained in datagram headers while they are transmitted.

The time taken for locating a remote service instance that is not already known in the remote repository is given by the response time of service requests issued to peer ND services. Hence, it compares to the response time of Web services access discussed in the previous section, since the ND service is implemented as a Web service. In particular, note that related SOAP messages are of low complexity in terms of the number of XML elements, and hence compares to the response time obtained with arrays of less than 20 elements.

In addition to the time taken for processing ND requests, the SLP server periodically looks-up for peer ND services. The process of locating peer ND services takes about 0.3ms per peer node in the network. In general, the overall performance of service retrieval using the ND service is dependent upon the specific network configuration and of the access patterns of nodes. In the case of quite static mobile nodes and/or infrastructure-based networking, the time taken for service location is not significant. On the other hand, a network of highly mobile nodes whose users access a variety of services over time may significantly impact upon performance, which we plan to investigate through simulation.

Our base prototype effectively supports the scenarios discussed in Section 3.1, which is in particular due to the fact that the Cybercar offers a wireless interface in the infrastructure-based mode through an out-door WLAN. Hence, interactions among devices within the Cybercar are in the ad hoc mode, while interactions with other nodes are in the infrastructure-based mode (see 4). However, further improvement to the ND service should be integrated for offering efficient service location in most situations. In particular, adequate prefetching and caching techniques must be devised.

6.4 Connector Customization

Connector customization to enforce quality of service over interactions is implemented according to the design presented in Section 5.3: the WSAMI middleware retrieves the required customizers and transparently handles the routing of the messages towards client/service via

the retrieved customizers. Implementation of connector customization is based on the handler mechanism that is defined in the JAX-RPC specification. Handlers are components that can plug in additional RPC processing behavior and enhance functionality of the RPC system (e.g., encryption and decryption, authentication and authorization).

Note that the proposed solution assumes that the nodes hosting the customizers retrieved on the client actually allow for resulting interactions between related services. This may not be assumed in general if the nodes are in the ad hoc mode (e.g., the nodes hosting the server-side remote customizers and the one hosting the invoked service may both be in the communication range of the client node but may not be in the communication range of each other). Such an issue will be addressed in our next prototype, through the integration of an ad hoc routing protocol.

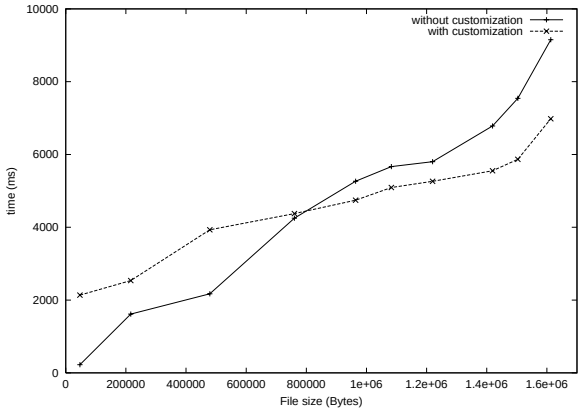

Figure 10: Customizer reducing image size versus absence of customization

By construction, the performance impact of customization relates to the performance of Web services accesses. In general, the overall cost of customization has to be balanced with the quality of service that it enables to achieve. For instance, customizers that filter the content transmitted over the network by reducing the messages size allow to both adapt the content to the target device, and to offer better response time to users for large content. Figure 10 gives the response time associated with getting a image file on a PDA from a service running on a laptop, using the 11MB WLAN in the ad hoc mode, according to the size of the original image file, with and without reducing the image size via customization. The corresponding customizer services are co-located with their associated end-point and the remote customizer reduces the image size using JMF (Java Media Framework) so that it fits the PDA's screen. In addition to making the necessary adaptation of the content for display on the PDA, results show that the connector customization further improves performance from the standpoint of response time for image files that are greater than 800KB (corresponding to a reduced image file of about 7KB), considering communication over an 11MB WLAN. Also, note that the current CSOAP implementation does not optimize calls to co-located services, and hence leads to response times higher than what can actually be achieved.

7 Conclusion

The vision of ambient intelligence is among today's most challenging topics for information technology. Realizing the vision means that consumers will be provided with universal and immediate access to available content and services, together with ways of effectively exploiting them. However, while base networking and hardware technologies are here to enable the vision to become a reality, there is still a long way to go before its full, robust realization. Open issues include provisioning: multi-modal user interfaces, software environments so that applications deployed in the computing space work effectively independently of the consumer's profile and location, network protocols for improved connectivity in any situation, and hardware for, e.g., enhanced autonomy of tiny-scale devices such as wearable computers. This paper has concentrated on one such issue that is supporting the development of ambient intelligence software systems.

Our solution primarily builds on results of service-oriented software engineering and architecture-based development of software systems, which have been proven successful for the development of distributed software systems: ambient intelligence applications are developed in terms of the composition of services that are defined through their abstract interfaces. The ambient intelligence requirement of enabling any-time, any-where access to applications from any terminal further leads to bind with related services instances at runtime, according to the environment in which the service is requested and in particular service instances that may be reached. Such a facility then requires a software technology that is pervasive enough for being able to rely on both consistent specification and availability of services in most environments, so as to actually support any-time, any-where discovery of service instances from abstract interfaces. This has led us to base our solution on the Web, and more specifically on the Web services architecture. Our main design objective was then to offer a solution that could be deployed in any environment, and effectively supported by mobile, resource-constrained devices.

We have thus introduced the XML-based WSAMI declarative language for the specification of Web services taking part in the realization of ambient intelligence applications (see § 4), together with associated core SOAP-based WSAMI middleware (see § 5). The language allows for dynamically retrieving instances of services matching a requested application by comparing only URIs of XML documents. Hence, the cost associated with the dynamic composition of applications, in terms of resource consumption (and in particular energy) is kept very low, which is mandatory for wireless devices. The language further allows for composing applications that guarantee security and performance properties, which we consider as prime requirements for the actual acceptance of ambient intelligence by consumers. Actual composition of applications at runtime relies on the core middleware, which amounts to supporting SOAP and to a naming&discovery service for the dynamic retrieval of service instances, both in the local and the wide area over WLAN, according to the network connectivity and associated cost.

Supporting the development of ambient intelligence or pervasive computing systems has given rise to extensive research over the last couple of years (see § 2), which has led to introduce a number of complex middleware-related services that place high demand on the underlying platform and hence limit deployment in most environments. Our contribution lies in the definition of a minimal middleware infrastructure for the actual dynamic composition of services,

i.e., a naming&discovery service in addition to SOAP, which allows for its wide deployment but also incurs minimal overhead in terms of resource consumption and response time. Additional middleware-related services may be exploited for increased quality of service, depending on the specific target application, but they do not have to be supported in all environments. Our solution resembles work in the area of service discovery, given the base support offered by the platform. However, results in the area target local area networks, while we target composition of services that may be retrieved both in the local and the wide area. In addition, by building upon the Web services architecture, availability of services is promoted.

This paper has more specifically introduced the core of our solution, that is, the WSAMI language and associated naming & discovery service for the development of ambient intelligence applications based on Web services. Compared to the state of the art related the Web services architecture, our work distinguishes itself by addressing the dynamic discovery and composition of Web services in the mobile context. This leads us to address the deployment of Web services (considering both client and server) on wireless, resource-constrained terminals, and supporting the dynamic composition of Web services in a way that enforces quality of service despite high variability in resources availability and dynamics of the network. Quality of service is being investigated as future extension of the Web services architecture, which shall lead to associated XML schemas for the specification of Web services. Such a specification can easily be integrated in the WSAMI language to replace our current solution, hence allowing for compatibility with the base Web services architecture. In the same way, the WSAMI language allows for integration with complementary XML languages for the specification of Web services choreography and orchestration. Our contribution lies in offering support for dealing with the mobile environment, further leading to support ubiquitous computing and networking using Web services.

We have implemented a demonstrator supporting typical ambient intelligence scenarios (see § 3) on top of the first prototype implementation of our environment (see § 6), which will serve getting feedback from consumers as part of the Ozone IST project [OZONE, 2003]. First experiments with our prototype are encouraging: (i) the development of ambient intelligence systems using our environment does not add any complexity compared to the one of Web services, (ii) performance of the resulting systems are comparable to the ones obtained with traditional middleware and allows for execution on wireless, resource-constrained devices, and the overhead related to the functions handling user's mobility (i.e., situation-sensitive composition relying on service discovery and connector customization) compares to the cost of base Web services access. We are currently working on complementing our core solution with support for off-line analysis of Web services so as to enforce robustness of the composed applications, regarding in particular behavioral matching with respect to both functional and non-functional properties. We are also investigating the exploitation of user and server profiles for the naming&discovery service, which will in particular enable an enhanced service selection process with respect to performance, and the integration of advanced prefetching techniques for enhancing response time. The current WSAMI middleware is targeted at one-hop networks, leading to a restricted form of ubiquitous networking. Part of our research is dedicated to enhancing WSAMI towards the comprehensive integration of Mobile Ad Hoc Networks (MANET), further leading to investigate middleware-related services for managing highly dynamic networks and related mobility-induced failures.

An issue that is often raised in the area of ambient intelligence is the one of dynamic adaptation to account for changes in resource availability at runtime (e.g., changing network connectivity, decreasing energy). A number of techniques but also dedicated middleware have been proposed in the literature to address it. This issue is partly addressed in our solution; it relies on the specification of adequate middleware-related services (e.g, specifying and exploiting a storage service that supports prefetching and caching). The core naming&discovery service further accounts for changes in the availability of service instances that may be accessed. However, adaptation remains an area for future work. It further needs assessing the benefits of dynamic adaptation against various factors such as the frequency of environmental changes over the duration of a given service access, and the resources that are consumed under adaptation versus in the absence of adaptation.

Acknowledgment

This work has received the support of the European Commission through the IST programme, as part of the Ozone project. The authors would further like to thank the anonymous reviewers for their useful comments.

References

[Aarts et al., 2001] Aarts, E., Harwig, R., and Schuurmans, M. (2001). Ambient intelligence. In *The Invisible Future: The Seamless Integration of Technology into Everyday Life*. McGraw-Hill Professional.

[Baggio et al., 2001] Baggio, A., Ballintijn, G., van Steen, M., and Tanenbaum, A. (2001). Efficient tracking of mobile objects in Globe. *The Computer Journal*, 44(5).

[Benatallah et al., 2002] Benatallah, B., Dumas, M., Fauvet, M.-C., and Rabhi, F. (2002). Towards patterns of Web services composition. In *Patterns and Skeletons for Parallel and Distributed Computing*. Springer.

[Bettstetter and Renner, 2000] Bettstetter, C. and Renner, C. (2000). A comparison of service discovery protocols and implementation of the service location protocol. In *Proceedings of the 6th EUNICE Open European Summer School: Innovative Internet Applications*.

[Blair et al., 2000] Blair, G., Blair, L., Issarny, V., Tuma, P., and Zarras, A. (2000). The role of software architecture in constraining adaptation in component-based middleware platforms. In *Proceedings of Middleware'00 – The ACM/IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*. LNCS 1795.

[Borriello and Want, 2000] Borriello, G. and Want, R. (2000). Embedded computation meets the World Wide Web. *Communications of the ACM*, 43(5).

[Boulkenafed and Issarny, 2003] Boulkenafed, M. and Issarny, V. (2003). A middleware service for mobile ad hoc data sharing, enhancing data availability. In *Proceedings of the*

4th ACM/IFIP/USENIX International Middleware Conference. Available at http://www-rocq.inria.fr/arles/doc/doc.html.

[Castro et al., 2001] Castro, P., Greenstein, B., Muntz, R., Bisdikian, C., Kermani, P., and Papadopouli, M. (2001). Locating application data across service discovery domains. In *Proceedings of ACM SIGMOBILE'01*.

[Cheng et al., 2002] Cheng, S., Garlan, D., Schmerl, B., Sousa, J., Spitznagel, B., Steenkiste, P., and Hu, N. (2002). Software architecture-based adaptation for pervasive systems. In *Proceedings of the International Conference on Architecture of Computing Systems: Trends in Network and Pervasive Computing*.

[Dertouzos, 1999] Dertouzos, M. L. (1999). The future of computing. *Scientific American*.

[Ekenstam et al., 2001] Ekenstam, T., Matheny, C., Reiher, P., and Popek, G. (2001). The Bengal database replication system. *Distributed and Parallel Databases*, 9(3).

[Esler et al., 1999] Esler, M., Hightower, J., Anderson, T., and Borriello, G. (1999). Next century challenges: Data-centric networking for invisible computing. In *Proceedings of MOBICOM'99*.

[Fox et al., 1998] Fox, A., Gribble, S. D., and Chawathe, Y. (1998). Adapting to network and client variation using active proxies: Lessons and perspectives. *Special Issue of IEEE Personal Communications on Adaptation*.

[Friedman, 2002] Friedman, R. (2002). Caching Web services in mobile ad hoc networks: Opportunities and challenges. In *Proceedings of the 2nd ACM Workshop on Principles of Mobile Computing (POMC)*.

[Fu et al., 2001] Fu, X., Shi, W., Akkerman, A., and Karamcheti, V. (2001). CANS: composable, adaptive network services infrastructure. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*.

[Garlan et al., 2002] Garlan, D., Siewiorek, D., Smailagic, A., and Steenkiste, P. (2002). Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2).

[Grace et al., 2003] Grace, P., Blair, G., and Samuel, S. (2003). Middleware awareness in mobile computing. In *Proceedings of the 1st International ICDCS Workshop on Mobile Computing Middleware*. Available at http://www.cs.ucl.ac.uk/staff/c.mascolo/mcm03.

[Issarny et al., 2002] Issarny, V., Kloukinas, C., and Zarras, A. (2002). Systematic aid for developing middleware architectures. *Communications of the ACM*, 45(6).

[IST Advisory Group, 2001] IST Advisory Group (2001). Scenarios for ambient intelligence in 2010. Technical report, ISTAG. Available at http://www.cordis.lu/ist/istag.htm.

[Joseph et al., 1995] Joseph, A., De Lespinasse, A., Tauber, J., Gifford, D., and Kaashoek, M. (1995). Rover: A toolkit for mobile information access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*.

[Kloukinas and Issarny, 2000] Kloukinas, C. and Issarny, V. (2000). Automating the composition of middleware configurations. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering*.

[Kon et al., 2002] Kon, F., Costa, F., Blair, G., and Campbell, R. (2002). The case for reflective middleware. *Communications of the ACM*, 45(6).

[Kubiatowicz et al., 2000] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. (2000). Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ASPLOS*.

[Kuenning and Popek, 1997] Kuenning, G. and Popek, G. (1997). Automated hoarding for mobile computers. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*.

[Lamanna et al., 2003] Lamanna, D. D., Skene, J., and Emmerich, W. (2003). SLAng: A language for defining service level agreements. In *Proceedings of the 4th ACM/IFIP/USENIX International Middleware Conference*. Available at http://www.cs.ucl.ac.uk/staff/W.Emmerich/publications/Middleware03/index.html.

[Liu and Issarny, 2004] Liu, J. and Issarny, V. (2004). Qos-aware service location in mobile ad hoc networks. In *Proceedings of the 5th IEEE International Conference on Mobile Data Management (MDM'04)*.

[Mascolo et al., 2001] Mascolo, C., Capra, L., Zachariadis, S., and Emmerich, W. (2001). XMIDDLE: a data-sharing middleware for mobile computing. *Wireless Personal Communications*, 21.

[McFarland and Wong, 2003] McFarland, B. and Wong, M. (2003). The family dynamics of 802.11. *ACM Queue – Tomorrow's Computing Today*, 1(3).

[Medvidovic and Taylor, 2000] Medvidovic, N. and Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1).

[Milojicic et al., 2001] Milojicic, D., Messaer, A., Bernadat, P., Greeberg, I., Spinczyk, O., Beuche, D., and Shroder-Preikschat, W. (2001). ψ - pervasive services infrastructure. In *Proceedings of TES'01*. LNCS 2193.

[Narayanan and McIlraith, 2002] Narayanan, S. and McIlraith, S. (2002). Simulation, verification and automated composition of Web services. In *Proceedings of the WWW'02 Conference*.

[Noble et al., 1997] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. (1997). Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*.

[OZONE, 2002] OZONE (2002). New technologies and services for emerging nomadic societies. Technical report, The OZONE Consortium. http://www.extra.research.philips.com/euprojects/ozone/.

[OZONE, 2003] OZONE (2003). Deliverable 9a: Applications scenarios. Technical report, The OZONE Consortium. http://www.extra.research.philips.com/euprojects/ozone/.

[Petersen et al., 1997] Petersen, K., Spreitzer, M., Terry, D., Theimer, M., and Demers, A. (1997). Flexible update propagation for weakly consistent replication. In *Proceedings of the 16th Symposium on Operating Systems Principles*.

[Poor et al., 2003] Poor, R., C.Bowman, and Auburn, C. B. (2003). Self-healing networks. *ACM Queue – Tomorrow's Computing Today*, 1(3).

[Ritter, 2003] Ritter, M. (2003). Mobility network systems. *ACM Queue – Tomorrow's Computing Today*, 1(3).

[Roman et al., 2000] Roman, G.-C., Picco, G., and Murphy, A. (2000). Software engineering for mobility: A roadmap. In *Proceedings of the 22nd International Conference on Software Engineering*.

[Sahai et al., 2002] Sahai, A., Durante, A., and Machiraju, V. (2002). Towards Automated SLA Management for Web Services. Technical Report HPL-2001-310R1, HP Laboratories Palo Alto. Available at http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.html.

[Sailhan and Issarny, 2003] Sailhan, F. and Issarny, V. (2003). Cooperative caching in ad hoc network. In *Proceedings of the 4th International Conference on Mobile Data Management (MDM)*. LNCS 2574.

[Sairamesh et al., 2002] Sairamesh, J., Goh, S., Stanoi, I., Li, C., and Padmanabhan, S. (2002). Self-managing, disconnected processes and mechanisms for mobile e-business. In *Proceedings of the 2nd International Workshop on Mobile Commerce*. Available at http://www.research.ibm.comcompsci/spotlight/ecommerce/ACMMobicomPaper2002.pdf.

[Sousa and Garlan, 2002] Sousa, J. P. and Garlan, D. (2002). Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*.

[Steinberg and Pasquale, 2002] Steinberg, J. and Pasquale, J. (2002). A Web middleware architecture for dynamic customization of content for wireless clients. In *Proceedings of the WWW'02 Conference*.

[Tartanoglu et al., 2003] Tartanoglu, F., Issarny, V., Romanovsky, A., and Levy, N. (2003). Dependability in the Web services architecture. In *Architecting Dependable Systems*. Springer Verlag, LNCS 2677. Available at http://www-rocq.inria.fr/arles/doc/doc.html.

[Terry and Ramasubramanian, 2003] Terry, D. and Ramasubramanian, V. (2003). Caching XML Web Services for mobility. *ACM Queue – Tomorrow's Computing Today*, 1(3).

[W3C, 2001] W3C (2001). Semantic web. Technical report, The World Wide Web Consortium. http://www.w3.org/2001/sw/.

[W3C, 2002] W3C (2002). Web services activity. Technical report, The World Wide Web Consortium. http://www.w3.org/2002/ws/.

[Weiser, 1993] Weiser, M. (1993). Some computer science problems in ubiquitous computing. *Communications of the ACM*, 36(7).

Appendix: WSAMI Schema

```
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www-rocq.inria.fr/arles/wsami"
targetNamespace="http://www-rocq.inria.fr/arles/wsami"
elementFormDefault="qualified" attributeFormDefault="unqualified" >

<xsd:element name="wsami">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Definition" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Definition">
  <xsd:complexType >
    <xsd:choice minOccurs="1" maxOccurs="1" >
       <xsd:element ref="Abstract" />
       <xsd:element ref="Service" />
       <xsd:element ref="Customizer"/>
    </xsd:choice >
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="targetNamespace" type="xsd:anyURI"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Abstract">
  <xsd:complexType >
    <xsd:sequence>
       <xsd:element name="Interface"  type="XMLDocumentType"/>
       <xsd:element name="Conversation" type="XMLDocumentType"  minOccurs="0"/>
```

```
      <xsd:element ref="ServiceQoS" minOccurs="0" />
      <xsd:element ref="ConnectorQoS" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" />
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="XMLDocumentType">
  <xsd:attribute name="hrefSchema" type="xsd:anyURI"/>
</xsd:complexType>

<xsd:element name="Service">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Abstract" type="XMLDocumentType"/>
      <xsd:element name="Concrete" type="XMLDocumentType"/>
      <xsd:element ref="Required" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Required">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ReqService" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element  name="ReqService">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Abstract" type="XMLDocumentType"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="optional" type="xsd:boolean" default="false"/>
    <xsd:attribute name="instance" type="xsd:boolean" default="false"/>
    <xsd:attribute name="multiple" type="xsd:boolean" default="false"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Customizer">
  <xsd:complexType>
```

```
    <xsd:sequence>
      <xsd:element ref="QoSCriterion" maxOccurs="unbounded"/>
      <xsd:element name="Local" type="XMLDocumentType"/>
      <xsd:element name="Remote" type="XMLDocumentType"/>
      <xsd:element name="Colocation" type="xsd:boolean" default="false"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ServiceQoS">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="QoSCriterion" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ConnectorQoS">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="QoSCriterion" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="QoSCriterion">
  <xsd:complexType>
    <xsd:attribute name="name" type="QoSType"/>
  </xsd:complexType>
</xsd:element>

<xsd:simpleType name="QoSType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="transactionalService"/>
    <xsd:enumeration value="security"/>
    <xsd:enumeration value="saveBandwidth"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```